IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
REQUEST FOR FILING APPLICATION Under Rule 53(a),(b)&(f)
(No Filing Fee or Declaration); RULE 53(f) NO DECLARATION

PATENT
APPLICATION

Asst. Commissioner for Patents
Washington, DC 20231

Atty. Dkt.	54355	MAG-005
	M# 253032	Client Ref

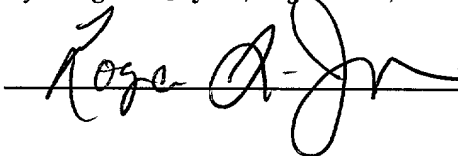Date: April 27, 1999

U.S. PTO
09/300405
04/27/99
JC135

Sir:

1. This is a Request for filing a new PATENT APPLICATION: ☐Design ☒ Utility, without a filing fee or Oath/Declaration, entitled:

2. Title: **Method for Storing Multiple Levels of Design Data in a Common Database**, for which enclosed are:

3. ☒ Abstract 1 page.   4.   **22**   Pages of specification and claims;

5. **30** claims   6.   ☒   Drawings: **04** sheets of informal on 8½x11; and

7. Form PTO 1449 and references cited therein.

8. This application is made by the following named inventors:

a.   Name: **Lukas P.P.P. VAN GINNEKEN**   Country of Citizenship: **The Netherlands**
     Residence (City):   **San Jose, California**
     Post Office Address: **4792 Country Lane**
                             San Jose, CA   95129

b.   Name: **Patrick R. GROENEVELD**   Country of Citizenship: **The Netherlands**
     Residence (City):   **San Jose, California**
     Post Office Address: **1431 Saratoga Avenue, Apt. 112**
                             San Jose, CA   95129

c.   Name: **Wilhelmus J.M. PHILIPSEN**   Country of Citizenship: **The Netherlands**
     Residence (City):   **Phoenix, Arizona**
     Post Office Address: **4337 East Sunrise Drive**
                             Phoenix, AZ   85044

1100 New York Avenue, N.W.
Ninth Floor, East Tower
Washington, D.C. 20005-3918
Tel: (650) 233-4552
Fax: (650) 233-4545

Pillsbury Madison & Sutro LLP

By: **Roger S. Joyner**, Reg. No. 36,176

54355/253032

METHOD FOR STORING MULTIPLE LEVELS OF

DESIGN DATA IN A COMMON DATBASE

BACKGROUND OF THE INVENTION

5   1. Field of the Invention

The present invention is directed to digital logic design systems. More particularly,

the invention is directed to automated digital logic synthesis and placement systems.

2. Background of the Related Art

10   Prior art computer aided design (CAD) systems for the design of integrated circuits

and the like assist in the design thereof by providing a user with a set of software tools running on a

computer. In the prior art, the process of designing an integrated circuit on a typical CAD system

was done in several discrete steps using different software tools.

First, a schematic diagram of the integrated circuit is entered interactively to

15   produce a digital representation of the integrated circuit elements and their interconnections. This

representation may initially be in a hardware description language such as Verilog and then

translated into a register transfer level (RTL) description in terms of pre-designed functional

blocks, such as memories and registers. This may take the form of a data structure called a net list.

Next, a logic compiler receives the net list and, using a component database, puts all

20   of the information necessary for layout, verification and simulation into object files whose formats

are optimized specifically for those functions.

Afterwards, a logic verifier checks the schematic for design errors, such as multiple

outputs connected together, overloaded signal paths, etc., and generates error indications if any

such design problems exist. In many cases, the IC designer improperly connected or improperly placed a physical item within one or more cells. In this case, these errors are flagged to the IC designer so that the layout cells may be fixed so that the layout cells perform their proper logical operation. Also, the verification process checks the hand-laid-out cells to determine if a plurality

5   of design rules have been observed. Design rules are provided to integrated circuit designers to ensure that a part can be manufactured with greater yield. Most design rules include hundreds of parameters and, for example, include pitch between metal lines, spacing between diffusion regions in the substrate, sizes of conductive regions to ensure proper contacting without electrical short circuiting, minimum widths of conductive regions, pad sizes, and the like. If a design rule violation

10   is identified, this violation is flagged to the IC designer so that the IC designer can properly correct the cells so that the cells are in accordance with the design rules.

Then, using a simulator the user of the CAD system prepares a list of vectors representing real input values to be applied to the simulation model of the integrated circuit. This representation is translated into a form which is best suited to simulation. This representation of

15   the integrated circuit is then operated upon by the simulator which produces numerical outputs analogous to the response of a real circuit with the same inputs applied. By viewing the simulation results, the user may then determine if the represented circuit will perform correctly when it is constructed. If not, he or she may re-edit the schematic of the integrated circuit, re-compile and re-simulate. This process is performed iteratively until the user is satisfied that the design of the

20   integrated circuit is correct.

Then, the human IC designer presents as input to a logic synthesis tool a cell library and a behavioral model. The behavioral circuit model is typically a file in memory which looks very similar to a computer program. The behavioral circuit model contains instructions which

2

define logically the operation of the integrated circuit. The logic synthesis tool receives as input the instructions from the RTL circuit model (i.e., Verilog or VHDL) and the library cells from the library. The synthesis tool maps the instructions from the behavioral circuit model to one or more logic cells from the library to transform the behavioral circuit model to a gate schematic net list of

5   interconnected cells. A gate schematic net list is a data base having interconnected logic cells which perform a logical function in accordance with the behavioral circuit model instructions. Once the gate schematic net list is formed, it is provided to a place and route tool.

The place and route tool is used to access the gate schematic net list and the library cells to position the cells of the gate schematic net list in a two-dimensional format within a surface

10   area of an integrated circuit die perimeter. The output of the place and route step is a two-dimensional physical design file which indicates the layout interconnection and two-dimensional IC physical arrangements of all gates/cells within the gate schematic net list.

According to the above prior art method, a separate internal data structure is used for each tool. This is because the tools are rarely if ever written by the same group; thus, the

15   internal database representation for each tools is likely to differ from that of the other tools. Also, the most appropriate database implementation for the integrated circuit depends on the phase of the design process in which it is being used. For example, linked lists are commonly used to store cells in a netlist because that is the most obvious solution for logic synthesis purposes. In contrast, a KD tree is a more appropriate database format for the place and route tool.

20   This is time-consuming and processor-intensive (circuit specifications must be translated from one database format to another and another during the development process), disk-intensive (multiple databases each specifying the same circuit in different forms must be stored)

and fragmented (tools cannot use the outputs of other tools, and a change to the circuit made by one tool is not reflected in the databases of the other tools).

SUMMARY OF THE INVENTION

5   The present invention has been made with the above problems of the prior art in mind, and a first object of the present invention is to provide a system for automated logic circuit design which is capable of storing and utilizing multiple levels of design data in a common database.

Another object of the present invention is to provide a system for automated logic
10   circuit design which eliminates the need for translation of circuit descriptions between different design tools.

A further object of the present invention is to provide a system for automated logic circuit design which allows the output of tools in the design suite to be used by other tools.

Yet another object of the present invention is to provide a system for automated
15   logic circuit design which allows design tools or the user to make area queries, i.e., a selection of a subset of objects based on their physical position, at various stages in the design process.

A still further object of the present invention is to provide a system for automated logic circuit design which permits the use of global simulation tools such as timing engines across all levels of design abstraction.

20   Another object of the present invention is to provide a system for automated logic circuit design which presents a unified model for timing, synthesis, placement and routing.

A further object of the present invention is to provide a system for automated logic circuit design which has high storage and run-time efficiency.

4

A still further object of the present invention is to provide a system for automated

logic circuit design which has a consistent and easy to use programming interface.

A still further object of the present invention is to provide a system for automated

logic circuit design which has an interface which is not dependent on other include files.

5   A further object of the present invention is to provide a system for automated logic

circuit design which uses an object-oriented C++ programming style.

The above objects are achieved according to an aspect of the invention by providing

an automated logic circuit design system which uses a common database to store design data at

different states of the design process, including data-flow graphs, netlists and layout descriptions.

10   In this way, the need to translate circuit descriptions between tools is eliminated, thus leading to

increased speed, flexibility and integration. The common database includes entities, models, cells,

pins, busses and nets. The data-flow graphs are stored as graphs, the nodes in a graph as cells, and

the edges as busses. Physical design data is available by storing the cells in a model in a KD tree.

This allows queries on cells in the netlist located in the layout within arbitrary areas.

15

5

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features, and advantages of the present invention are better

understood by reading the following detailed description of the preferred embodiment, taken in

conjunction with the accompanying drawings, in which:

5   FIGURES 1 - 3 are dataflow diagrams of a circuit structure according to a preferred

embodiment of the present invention;

FIGURE 4 is a block diagram of the structure of a data model according to the

present invention; and

FIGURE 5 is a diagram of the partitioning of a chip in correspondence with the data

10   model.

DETAILED DESCRIPTION OF THE

PRESENTLY PREFERRED EXEMPLARY EMBODIMENT

First, given a Verilog description of a circuit to be developed, the Verilog must be

15   parsed to generate a data flow graph suitable for implementation in the data model. RTL parsers

known in the art are preferably used for this purpose. The output from the RTL parser is a Verilog

parse tree which is used to generate the data flow graph. Although well-known in the art, the

structure of the parse tree is relatively complicated and, since detailed knowledge of it is not

necessary for an understanding of the present invention, further description of the parse tree will be

20   omitted for simplicity and brevity.

FIG. 1 shows an example of translation of the Verilog source code

```
always @(posedge clk)
begin
        out = in1 + in2;
25      if (c)
                out = in3;
```

```
end
```

into data flow elements. Here, in0, in1, in2, c and clk are input ports of an Entity (described below) and out is an output port of the Entity. An adder (an example of a Cell as described bellow) adds

5   the values at Ports in0 and in1 and supplies the result to a merge block (another example of a Cell). If the value at Port c represents a logical true, the merge block supplies the value at Port in2 to a delay block (again, a Cell); if the value at Port c represents a logical false, the merge block supplies the output of the adder to the delay block. On the positive-going edge of the signal at Port clk, the delay block provides the value on its input to the output port out. The data flow graph having been

10   generated, it may then be stored in the data model.

Before describing the data model in more detail, a few more examples are in order.

FIG. 2 shows an example of the data graph resulting from the Verilog code

```
       if (c1) begin
             out = in0;
15     end else begin
             if (c2) begin
                   out = in1
             end else begin
                   out = in2
20           end
       end
```

Here, if the value at input Port c1 is a logical true, the branch module connected to input Port in0 is enabled and provides its output to a merge module which is also enabled when c1 is true. If c1 is

25   not true, the branch modules connected to Ports in1 and in2 are enabled to provide their outputs to other branch modules. One of the modules in this second tier is enabled when the value at input Port c2 is true and provides its output to another merge block. The other of the modules in the second tier is enabled when c2 is false and provides its output to the other merge block. Depending on the value of c2, one of those outputs is provided to the first merge block, and depending on the

30   value of c1, one of the output is provided to the output port out via the delay element.

Finally, repetitive structures such as for, while and forever loops can be implemented in the data flow graph. FIG. 3 shows an example of the data flow graph implementation of the Verilog code

```
        integer i;
5       for (i=0; i<5; i++)
        begin
            a = a-4
        end
```

10   Here, an entry node initializes a loop index with the value 0, adds 1 to the index and checks to see if the index is less than 5. If so, an exit node loops back for another pass. In tandem with this loop, an input variable a is received through another entry node and 4 is subtracted from it on each pass through the loop. When looping ceases because the index has reached 5, the current value of the input variable is presented at the exit node.

15   Certain optimizations can be performed on the data flow graph. For example, in the above loop structure the loop can be unrolled. That is, the graph portion representing the body of the loop can be replicated five times and the graph portion representing the loop index can be eliminated. For timing estimations and the like, virtual loop unrolling can be performed by estimating the number of iterations through the loop and using that number as a multiplier in delay

20   calculations; the actual circuit replications can be done later.

            Once the Verilog source is converted to a data flow graph, it can be implemented in the data model. Preferably, the data model is implemented using the C++ programming language or a similar object-oriented language. Since the construction, accessing and destruction of objects in such languages is well-known in the art, examples of specific commands for performing these

25   operations will be omitted for brevity.

8

The topmost object in the data model 10 (shown in FIG. 4) is the Root object 20. The Root object 20 owns all other objects 30-70 and serves as a base to which everything else is attached. Also, the root 20 accommodates global attributes which are shared by all objects 20-70.

At the next level of the data model 10 is the Library object 30. Library objects 30 are used to organize entities 40. The preferred embodiment of the present invention includes at least two Libraries 30. The first stores information on the technology library to which the circuit under development will be mapped, and the second stores information on the circuit itself.

Next is the Entity object 40. An Entity 40 defines the interface of a design; that is, the set of ports 52 that the Entity 40 has. An Entity 40 may own Port objects 52. A Port 52 is used to represent the pin-out of an entity 40. Ports 52 cannot be individually created and destroyed, and can only be created when an Entity 40 is created. Each Port 52 has a direction (in, out, in/out) which is determined upon creation of the Entity 40. This rigidity promotes consistency between the Entity 40, its Models 50 and the Cells 62 bound to those Models 50.

As noted above, Entities 40 own Models 50. A Model 50 defines an implementation of an Entity 40; thus, multiple Models 50 within an Entity 40 represent different implementations of that Entity 40. Generally, these Models 50 are functionally identical to one another. For example, an Entity 40 in a technology Library 30 may have several Models 50 defining various eight bit adder cells having different power levels. Similarly, an Entity 40 in a target Library 30 may have several Models 50 which respectively define an abstract logic representation of a circuit, a gate-level implementation of it, a uniquefied representation, etc. The contents of each Model 50 is a net list of Nets 60, Cells 62 and Model Pins 64. All Models 50 and the Entity 40 have the same number of Pins 64 and Ports 52, and the Ports 52 have the same direction in the Entity 40 and over

9

all Models 50; thus, it is relatively easy to replace one Model 50 with another from the same Entity 40.

Below the Models 50 are Cell objects 62. A Cell 62 represents a section of logic. Cells 62 may be primitive cells or non-primitive cells. Primitive Cells 62 have a predefined functionality associated with them. Preferably, the primitive Cells 62 include the following types:

CELL_AND – unlimited fan-in AND gate;

CELL_XOR – unlimited fan-in OR gate;

CELL_TRI – tri-state buffer

CELL_REG – sequential element

CELL_DC – don't care set

CELL_BREAK – break point cell; used to implement a "don't touch"; and

CELL_ONE – a constant one; an inverted or bubbled version is used for a constant zero.

In contrast to primitive Cells, the functionality of non-primitive Cells is defined by technology Models 50 to which they are bound. That is, a Cell 62 may describe a portion of the circuit under development and belong to a Model 50 in a target Library 30. However, it will be associated with (preferably by pointing through a cell type attribute or the like) a Model 50 in a technology library 30 which defines its functionality and general characteristics.

Non-primitive Cells 62 may be created as bound Cells; alternatively, they may be created as unbound Cells and later bound to a Model 50. This may be done by specifying the Cell's name; by specifying pin-to-pin correspondence vectors; and by binding the Cell 62 to an undefined Model 50 and later matching the Model 50 to an actual one. Additionally, a bound Cell 62 can be rebound to a different Model 50 within the same Entity 40.

10

Each Cell 62 includes a number of parameters called members which specify certain features of the Cell 62. These include the cell's name, a pointer to the technology Model 50 to which it is bound, a list of Pins 64 which it owns, its parent Entity 40, and coordinates of the Cell 62 within the chip layout.

5   Net objects 60 make connections between pins. The pins may be Model pins 64 or Cell pins 70. A Net 60 does not own Pins 64 and 70, and deleting the Net 60 will leave the pins 64 and 70 disconnected. Pins 64 and 70 may be grouped into Busses 80 (in fact, every variable in the Verilog code will be represented as a Bus). Since Pins 64 and 70 are the most common object in almost any circuit representation, it is important to reduce the amount of storage for each Pin 64

10   and 70 as much as possible while maintaining easy accessibility. For this reason, Pins 64 and 70 are preferably stored in small arrays and associated with indices.

        Nets 60 also have members, such as the Net's name, a list of Pins 64 and 70 which it connects, and a list of rectangles through which it passes in the placement layout. Pin members include the Pin's name, the Model 50 or Cell 62 to which it belongs, and the Net 60 to which it is

15   connected.

        Each object 20-70 may have a number of attributes. Each attribute has a name and a value of a type int, short, float, double, char* and void*. One example of an object attribute is an inversion attribute or "bubble" which specifies whether a Cell input or output (or Net 60) is asserted high or low. Other examples of object attributes are object name, firing information,

20   references to the Verilog code defining the object, etc.

        Iterators are procedures used to access objects within the data model. As is known in the art, an iterator traverses the model and each time it is called, returns a pointer to the next object of a particular type. For example, a Model iterator would, when successively called, return

11

pointers to each Model 50 within the data model. The preferred embodiment of the present

invention provides "safe" and "unsafe" iterators, where unsafe iterators return all objects of the

specified type, even if they have been added during the iteration process, and safe iterators omit

objects added during the iteration. In this way, although the safe iterators are slightly slower than

5   their unsafe counterparts, they can avoid program crashes, errors and exceptions, and other

undesirable outcomes.

Before synthesis and timing can take place it is often necessary to uniquefy the data

model. This involves binding each Cell 62 to its own individual technology Model 50. This

simplifies the synthesis process in that changes made to one technology Model 50 will affect only

10   the Cell 62 which is bound to it, and no others. Also, after uniquefication it is possible to traverse

the data model both up and down, since each object has a unique parent and child. Typically,

uniquefication is done by making a copy of a technology Model 50 for each Cell 62 which is bound

to it and associating one of the cells 62 to each copy.

After the data model has been uniquefied, it may be ungrouped, i.e., macro-level

15   cells can be replaced with their primitive components. Alternatively, processes may handle the

data model with virtual ungrouping by "looking through" the macro-level cells to process their

primitive cell constituents.

With this understanding of the structure of the data model in mind, implementation

of a Verilog-derived data flow graph in the data model will now be described. For each module in

20   the Verilog description there will be one Entity 40 and one Model 50 (hereinafter collectively

referred to as a graph). The ports for the Entity 40 correspond to the ports in the Verilog module.

Ports 52 in the graph have a bit width, and there will be a separate Pin 64 and Net 60 (the group of

Nets 60 for the Port 52 forming a Bus) in the graph for each Verilog port.

For each node in the Verilog module, a Cell 62 will be made in the graph. Initially the Cells 62 will be unbound. As described above, given the Cell type and the Pins 70 of the Cell 62, a Model 50 for the Cell 62 to be bound can be generated later.

Each Model 50 is preferably implemented as a KD tree as follows. First, the circuit

5   under development is divided into a number of sections each corresponding to a rectangular section 100 of the available chip area as shown in FIG. 5. The partitioning of the circuit can be directed by the user; however, it is preferably automatically done by the system so that the circuit is evenly distributed over the entire chip area. Each node or leaf 210 of the KD tree 200 shown in FIG. 6 corresponds to a cutline 110 of the rectangles 100 and may have appended thereto a linked list 220

10   of all cells 62 which lie on that cutline 110. Non-leaf nodes 210 in the KD tree 200 each have two child nodes 210, with the left child 210 corresponding to the region of the chip on one side of the cutline 110 and the right child 210 corresponding to the region of the chip on the other side of the cutline 110. Similarly, the child nodes 210 may have linked lists 220 of cells on their cutlines 110 and child nodes 210 of their own.

15   It should be noted that the leaf nodes 210 will contain most of the circuit information, since the non-leaf nodes 210 will only have information on those cells touching their corresponding cutline.

As noted above, the initial distribution of Cells 62 over the chip area is preferably done automatically by the system and in that case may be done through the use of various

20   algorithms which will readily suggest themselves to those skilled in the art. The result of this process is a model with mostly logical information on its constituent elements but with a coarse framework of physical placement and routing information, e.g., cell areas, initial placements, etc. In later steps of the development process described below, the physical information will be refined

13

and augmented within the original data model. In this way, the addition of rough physical layout information to the initial logical description enables the smooth transition of the circuit through the development process, thereby enabling sharing of tool outputs, use of common diagnostics and the like.

5   Further, once RTL synthesis is complete and the data model is flattened, it may be copied and used as a baseline for formal verification and the like. Since a common model structure is used, there is no need to translate the pre-logic synthesis version of the circuit into a format suitable for use by the verification tool.

As the development process progresses, the KD tree 200 may become unbalanced

10   due to an excessive number of additions or deletions in one area, or due to poor initial distribution. This can be compensated for by manual rebalancing by the user or by a user-initiated procedure, but preferably is done automatically by the system.

Once the data model has been constructed in this way, it may be used for both logic synthesis, i.e., gate-level implementation, etc., and physical synthesis, i.e., placing and routing.

15   This is because the data model includes all of the information necessary for logical synthesis operations, i.e., cell functionality, net connections, etc., as well as all information necessary for physical synthesis operations, i.e., areas, physical positions, etc.

Another advantage of the data model arises from its correspondence with the actual physical chip layout. Since each node of the KD tree 200 corresponds to a cutline 110 and has

20   associated with it the cells on the cutline and information on where its child nodes are within the chip area, portions of the circuit in specific physical areas can be queried, tested and manipulated without the need to read the entire data model into active memory from disk storage, as is the case with prior art net lists. For example, assuming a user wanted to work with only the lower right

14

hand corner of the chip, the system could traverse the KD tree to reach the topmost node corresponding to that area. Then, that node, its children, netlists and the like would be read into active memory from disk and manipulated. The user may even be able to manually direct placement of cutlines 110 at certain points to frame a particular area of interest. The system may then adjust the KD tree accordingly to accommodate the new arrangement. This area query technique is possible whether the circuit is in its final placement and routing stages or fresh from Verilog synthesis.

Although only a portion of the entire data model need be read into memory, the complete set of Nets 60 is typically maintained in memory. This is because the Nets 60 are necessary for purposes such as delay estimation and the like that are performed frequently, and it is easier to retain all Nets 60 in memory rather than repeatedly read them into memory. Thus, once a specific area has been designated for querying, the Nets 60 corresponding to that area must be identified. This is done by identifying the Nets 60 connected to each of the Pins 64, 70 within the selected area. The remaining Nets 60 can be eliminated from consideration during the area query. Nets 60 which have some, but not all, Pins 64, 70 within the query area can have the missing pins represented by a stub pin. Finally, Nets 60 which have all of their pins within the query area can be handled as are other objects within the selected area.

Further, during the area query process, Nets 60 which are entirely contained within the selected area can be optimized out or otherwise modified; however those nets having portions outside the query area, i.e., those with stub pins, cannot, since the effect of modification of elimination of these Nets 60 on the remaining circuit portions is unpredictable.

Further, since the logical and physical aspects of the circuit are integrated into a single data model from the start, deviations from the classic logical synthesis/physical synthesis

partition can be made. For example, the inclusion of buffers for load handling and timing purposes is normally done as part of the logical synthesis process; however, using a common data model for all aspects of the development process allows the placement of buffers to be delayed until later during the placement process, when layout information is more definite and precise.

5   The above description of the preferred embodiment of the present invention has been given for purposes of illustration only, and variations thereof will be readily apparent to those skilled in the art. For example, although Verilog has been used as the preferred language for initial input of the circuit under development, other appropriate hardware description languages may of course be used. Also, although implementation of the data model using object-oriented C++ techniques has

10   been disclosed, other programming languages and paradigms may also be workable. Similarly, alternative object hierarchies may be used. Such variations fall within the scope of the present invention. Thus, the scope of the present invention should be limited only by the appended claims.

WHAT IS CLAIMED IS:

1. A method of digital circuit development comprising:

processing a description of the circuit in a computer system to generate electrical signals representative of a data model including information on logical parameters of the circuit

5 and information on physical parameters of the circuit;

causing the computer system to use the data model to synthesize logic elements in accordance with the logical parameters therein; and

causing the computer system to use the data model to generate physical placement information for the logic elements in accordance with the physical parameters therein.

10

2. The method of claim 1, wherein the description is in a hardware description language.

3. The method of claim 1, wherein the data model is a hierarchy of data objects.

15

4. The method of claim 3, wherein:

the hierarchy includes at least one library object which owns an entity object;

the entity object owns at least one model object; and

the model object is a functional representation of the circuit.

20

5. The method of claim 4, wherein:

the hierarchy includes a plurality of library objects;

at least one of the plurality of library object is a technology library object; and

17

at least one of the plurality of library objects is a target library object.

6. The method of claim 4, wherein:

the entity object owns a plurality of model objects;

each of the model objects is a different representation of the circuit; and

the model objects are functionally equivalent to one another.

7. The method of claim 4, wherein the at least one model object is implemented as a KD tree.

8. The method of claim 1, wherein the data model includes a representation of the circuit as a KD tree.

9. The method of claim 8, wherein each node of the KD tree corresponds to a physical attribute of a physical arrangement of the circuit.

10. The method of claim 9, wherein the physical attribute is a location of a portion of the circuit corresponding to the node within a chip.

11. The method of claim 9, wherein each node of the KD tree corresponds to a cutline of a semiconductor chip.

18

12. The method of claim 11, wherein at least one node of the KD tree has a cell associated therewith.

13. The method of claim 11, wherein at least one node of the KD tree has two child

5   nodes, the child nodes corresponding to portions of the circuit on opposite sides of the cutline.

14. The method of claim 1, wherein:

the data model includes a plurality of cells, each cell corresponding to a portion of

the circuit;

10   the logical parameters include functionality of the cells; and

the physical parameters include at least one of physical area and physical placement

of the cell.

15. The method of claim 14, wherein:

15   the physical parameters include physical placement of the cell; and

processing the circuit description includes a step of assigning arbitrary physical

placement parameters to the cells.

16. The method of claim 15, wherein causing the computer system to generate

20   physical placement information includes causing the computer to modify the arbitrary physical

placement parameters.

17. The method of claim 1, further comprising the step of using a common tool to evaluate the logical information and to evaluate the physical placement information.

18. The method of claim 17, wherein the common tool is a timing simulator.

5

19. The method of claim 1, further comprising:

storing the data model in the computer system;

specifying a physical area of the circuit less than the entire circuit area; and

retrieving a portion of the data model corresponding to the physical area while not

10   retrieving portions of the data model not corresponding to the physical area.

20. The method of claim 1, further comprising:

making a copy of the data model; and

using the copy to formally validate the circuit.

15

21. A computer-readable medium storing a data structure for representing digital circuits, wherein:

the data structure includes a root containing at least one library;

each library contains at least one entity;

20   each entity contains at least one model and at least one port;

each model contains at least one cell, at least one net and at least one pin; and

each cell contains at least one pin.

20

22. The computer-readable medium of claim 21, wherein the data structure includes both logical synthesis information and physical placement information.

23. The computer-readable medium of claim 21, wherein the library includes a

5   technology library and a circuit library.

24. The computer-readable medium of claim 21, wherein each entity is representative of a circuit.

10   25. The computer-readable medium of claim 24, wherein each model is representative of an implementation of the circuit represented by the entity which contains the model.

26. The computer-readable medium of claim 25, wherein:

15   at least one entity contains a plurality of models; and

the plurality of models are each representative of a different implementation of the circuit represented by the entity which contains the model.

27. The computer-readable medium of claim 21, comprising:

20   a plurality of cells;

wherein at least one of the plurality of cells is a primitive cell, and

at least one of the plurality of cells is a non-primitive cell.

28. The computer-readable medium of claim 21, wherein each model is represented in the data structure as a KD tree.

29. The computer-readable medium of claim 28, wherein each non-terminal node of the KD tree corresponds to a core cutline.

30. The computer-readable medium of claim 29, wherein each of the non-terminal KD tree nodes includes cells touching its corresponding cutline.
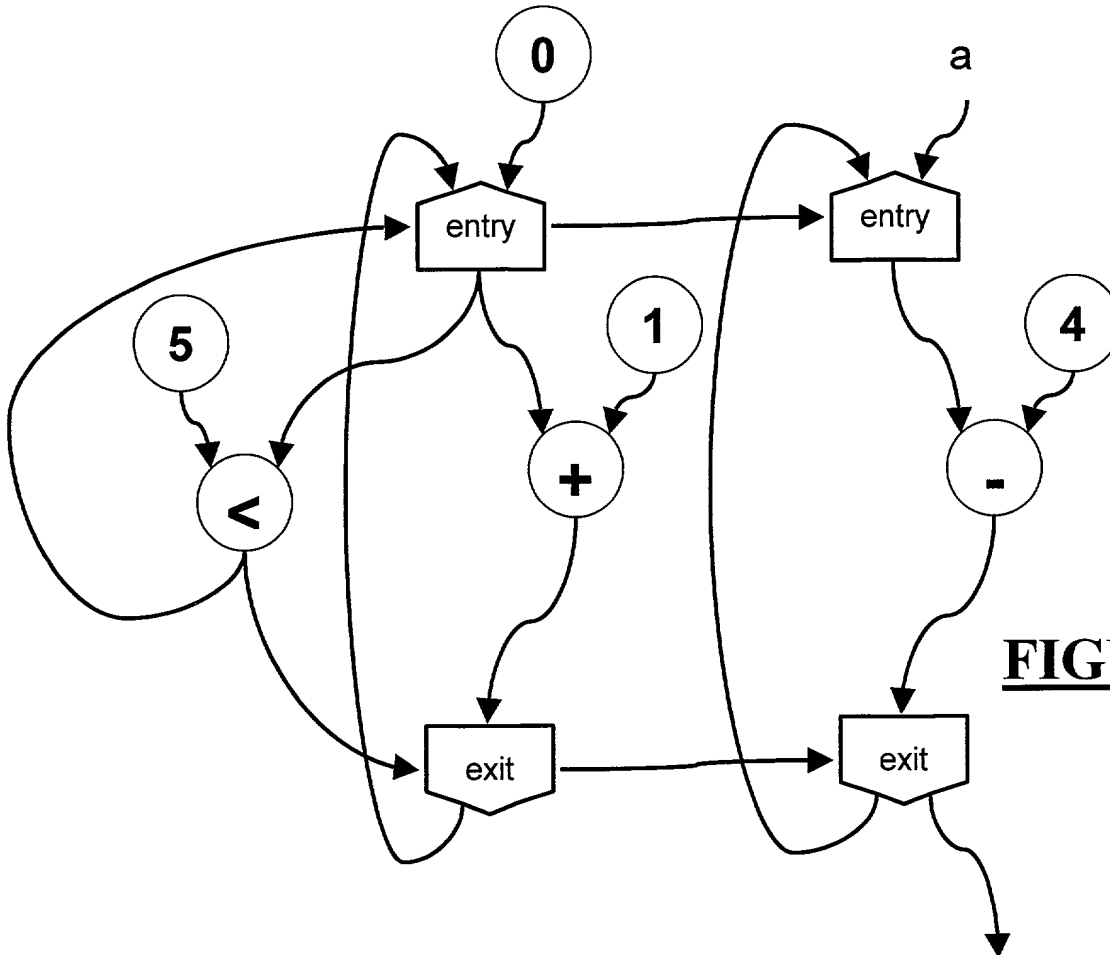
ABSTRACT OF THE DISCLOSURE

An automated logic circuit design system uses a common database to store design data at different states of the design process, including data-flow graphs, netlists and layout descriptions. In this way, the need to translate circuit descriptions between tools is eliminated, thus 5 leading to increased speed, flexibility and integration. The common database includes entities, models, cells, pins, busses and nets. The data-flow graphs are stored as graphs, the nodes in a graph as cells, and the edges as busses. Physical design data is available by storing the cells in a model in a KD tree. This allows queries on cells in the netlist located in the layout within arbitrary areas.

FIGURE 1

FIGURE 3

in0   in1   in2   in3   in4

branch ← branch ← branch

branch ← branch

merge

merge

@

out.

FIGURE 2

20

ROOT
(workspace or context)

10

30

30

LIB

LIB
(library)

40

40

ENTITY
(definition box or design)

ENTITY

50

MODEL
(proto box or network)

MODEL

PORT
(definition pin)

PORT

52

52

FIG. 4

62

NET

NET

CELL
(Usage box)

CELL

PIN
(proto pin)

PIN

62

64

64

60

62

PIN
(Usage pin)

PIN

70

FIGURE 5

100

110

62

100

62

110

62

100

100

62

110

62

110

110

100

100

110

62

100

200

FIGURE 6